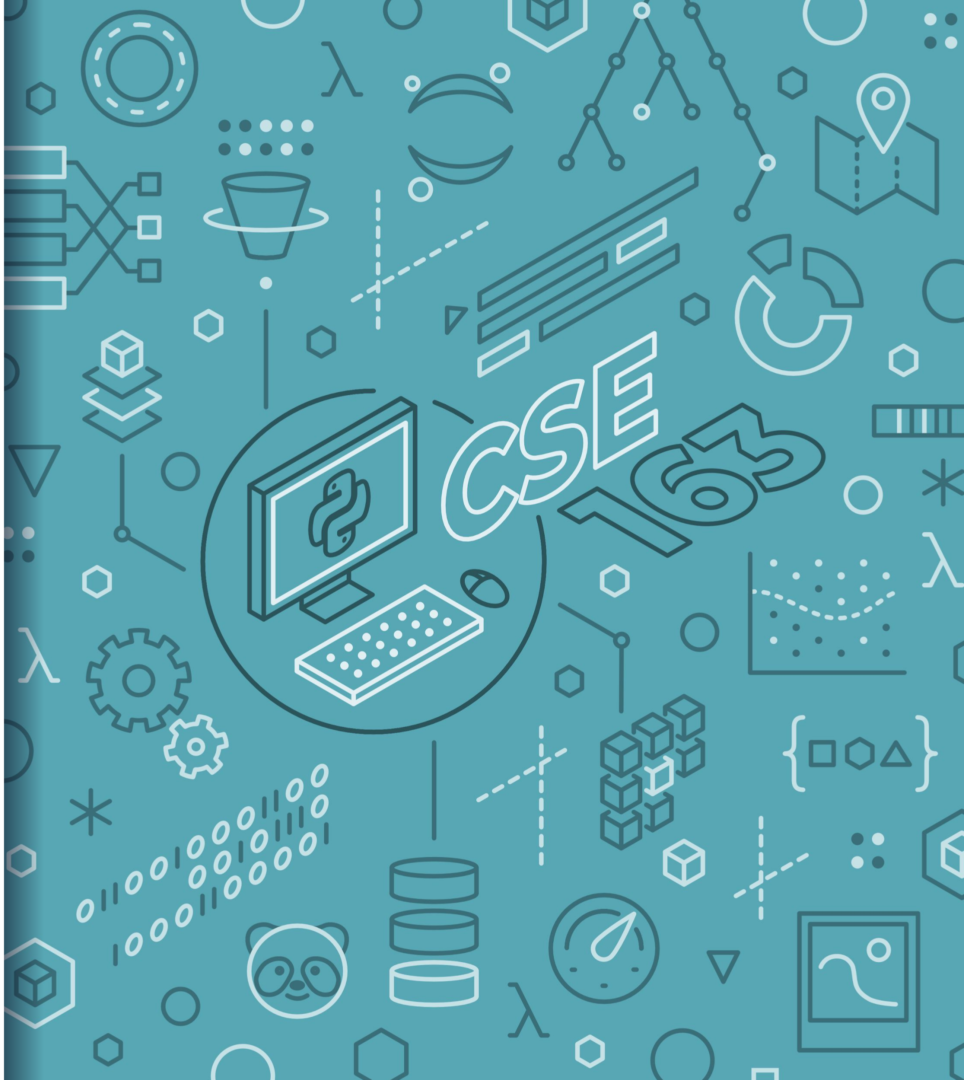
CSE 163

Joins / Spatial Indices

Hunter Schafer



Matplotlib

- Last time we showed how to plot 2 graphs on the same plot
- Terminology is a bit important so I wanted to cover that again

There are two fundamental concepts for matplotlib

- Figure (canvas to draw on / entire picture)
- Axis (an individual plot inside the figure)

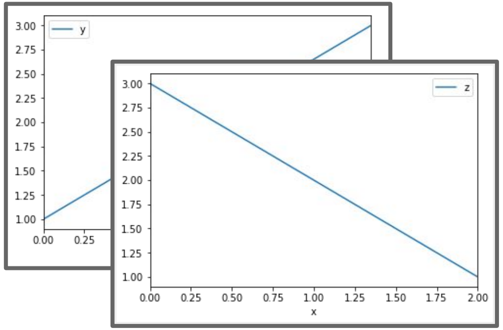
The subplots method conveniently returns a new figure and axis

```
fig, ax = plt.subplots(1)
```

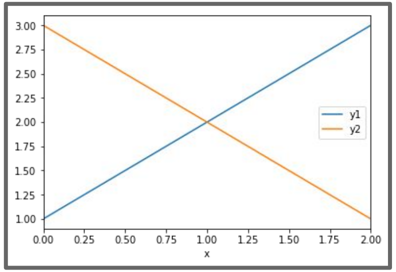
Matplotlib Example

x	y	z
1	1	3
2	2	2
3	3	1

```
df.plot(x='x', y='y')
df.plot(x='x', y='z')
plt.show()
```

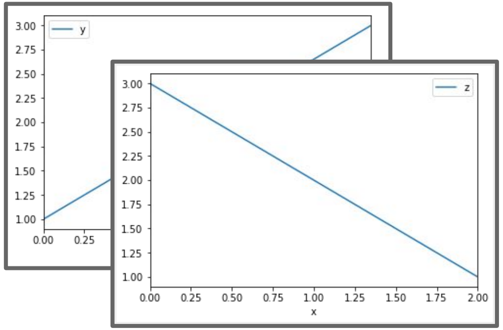


```
fig, ax = plt.subplots(1)
df.plot(x='x', y='y', ax=ax)
df.plot(x='x', y='z', ax=ax)
fig.show()
```

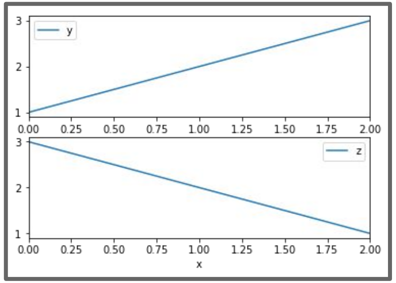


Matplotlib Example

```
df.plot(x='x', y='y')
df.plot(x='x', y='z')
plt.show()
```



```
fig, [ax1, ax2] =
    plt.subplots(2)
df.plot(x='x', y='y', ax=ax1)
df.plot(x='x', y='z', ax=ax2)
fig.show()
```



x	y	z
1	1	3
2	2	2
3	3	1

Dissolve

- Exactly the same as a groupby for the the regular columns
- For the geometry columns, overlays all of the geometries
- Options for `aggfunc`
  - `‘first’`
  - `‘last’`
  - `‘min’`
  - `‘max’`
  - `‘sum’`
  - `‘mean’`
  - `‘median’`

Colab

Separated Data

Imagine that your data is split into multiple DataFrames and you want to combine them.

tas

ta_name	ta_id
Joely	1
Dylan	2
Nicole	3

grading

grader_id	student_name
2	Hunter
3	Erika
1	Josh
3	Erik

How can I print out the grading assignments?

Combining Data

Basic Idea, write code to do something like (not real code)

```
for t in tas:
  for g in grading:
    if t['ta_id'] == g['grader_id']:
        print(t, g)
```

This is called a join since we are joining the tables together

- Find all combinations of rows that “line up” based on a value

This is such a common task, there is a method we can call to do this

Pandas Join

```
tas.merge(grading, left_on='ta_id',
          right_on='grader_id')
```

ta_name	ta_id
Joely	1
Dylan	2
Nicole	3

grader_id	student_name
2	Hunter
3	Erika
1	Josh
3	Erik

ta_name	ta_id	g_id	s_name
Joely	1	1	Josh
Dylan	2	2	Hunter
Nicole	3	3	Erika
Nicole	3	3	Erik

Pandas Join

```
tas.merge(grading, left_on='ta_id',
            right_on='grader_id')
```

ta_name	ta_id
Joely	1
Nicole	3

grader_id	student_name
2	Hunter
3	Erika
3	Josh
3	Erik

ta_name	ta_id	g_id	s_name
Nicole	3	3	Josh
Nicole	3	3	Erika
Nicole	3	3	Erik

Type of Join

There are interesting questions of what happens if there are rows that don't "line up". Different type of joins differ in how to handle this case.

Types of Joins

```
left.merge(right, how='type')
```

- **Inner** (default): Both values must be present
- **Left**: If a value from left has no match, add NaNs
- **Right:** If a value from right has no match, add NaNs
- **Outer:** If a value from either table has no match, add NaNs

Inner Join

```
tas.merge(grading, left_on='ta_id',
          right_on='grader_id')
```

ta_name	ta_id
Joely	1
Nicole	3

grader_id	student_name
2	Hunter
3	Erika
3	Josh
3	Erik

ta_name	ta_id	g_id	s_name
Nicole	3	3	Josh
Nicole	3	3	Erika
Nicole	3	3	Erik

Left Join

```
tas.merge(grading, left_on='ta_id',
          right_on='grader_id', how='left')
```

ta_name	ta_id
Joely	1
Nicole	3

grader_id	student_name
2	Hunter
3	Erika
3	Josh
3	Erik

ta_name	ta_id	g_id	s_name
Joely	1	NaN	NaN
Nicole	3	3	Josh
Nicole	3	3	Erika
Nicole	3	3	Erik

Right Join

```
tas.merge(grading, left_on='ta_id',
          right_on='grader_id', how='right')
```

ta_name	ta_id
Joely	1
Nicole	3

grader_id	student_name
2	Hunter
3	Erika
3	Josh
3	Erik

ta_name	ta_id	g_id	s_name
Nicole	3	3	Josh
Nicole	3	3	Erika
Nicole	3	3	Erik
NaN	NaN	2	Hunter

Outer Join

```
tas.merge(grading, left_on='ta_id',
          right_on='grader_id', how='outer')
```

ta_name	ta_id
Joely	1
Nicole	3

grader_id	student_name
2	Hunter
3	Erika
3	Josh
3	Erik

ta_name	ta_id	g_id	s_name
Joely	1	NaN	NaN
Nicole	3	3	Josh
Nicole	3	3	Erika
Nicole	3	3	Erik
NaN	NaN	2	Hunter

Geospatial Join

Colab

Goals

- Plot multiple layers on a map
- Find all the states that intersect the path of the hurricane

Notes

- Plot layers by plotting on the same axes
- Need to use a spatial join to join on geo-spatial features

Spatial Joins

Very similar to plain old joins (how)

- Same distinction between inner, left, right
- Find all pairs of matches

The key difference are you match by geo-spatial relation (op)

- Most commonly will just use `op=’intersects’`, but there are other ways to determine a match

```
geopandas.sjoin(mainland, florence, op='intersects')
```

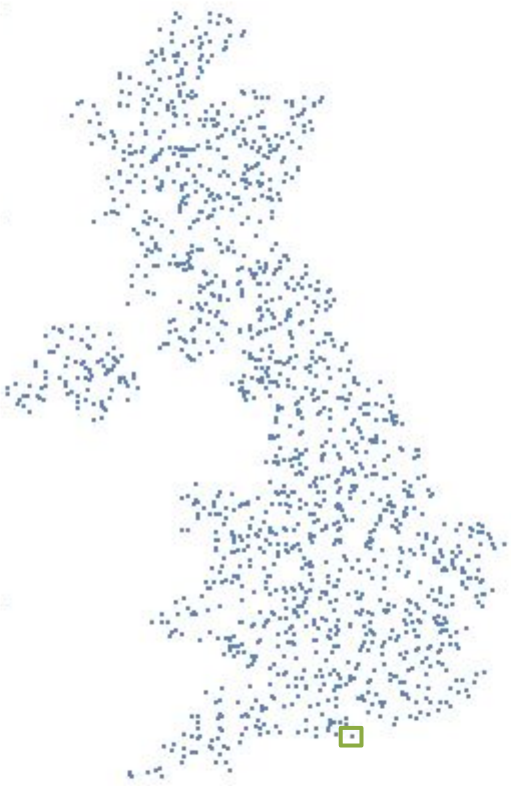
Spatial Join - Points

Imagine we had a dataset of where people live in England.

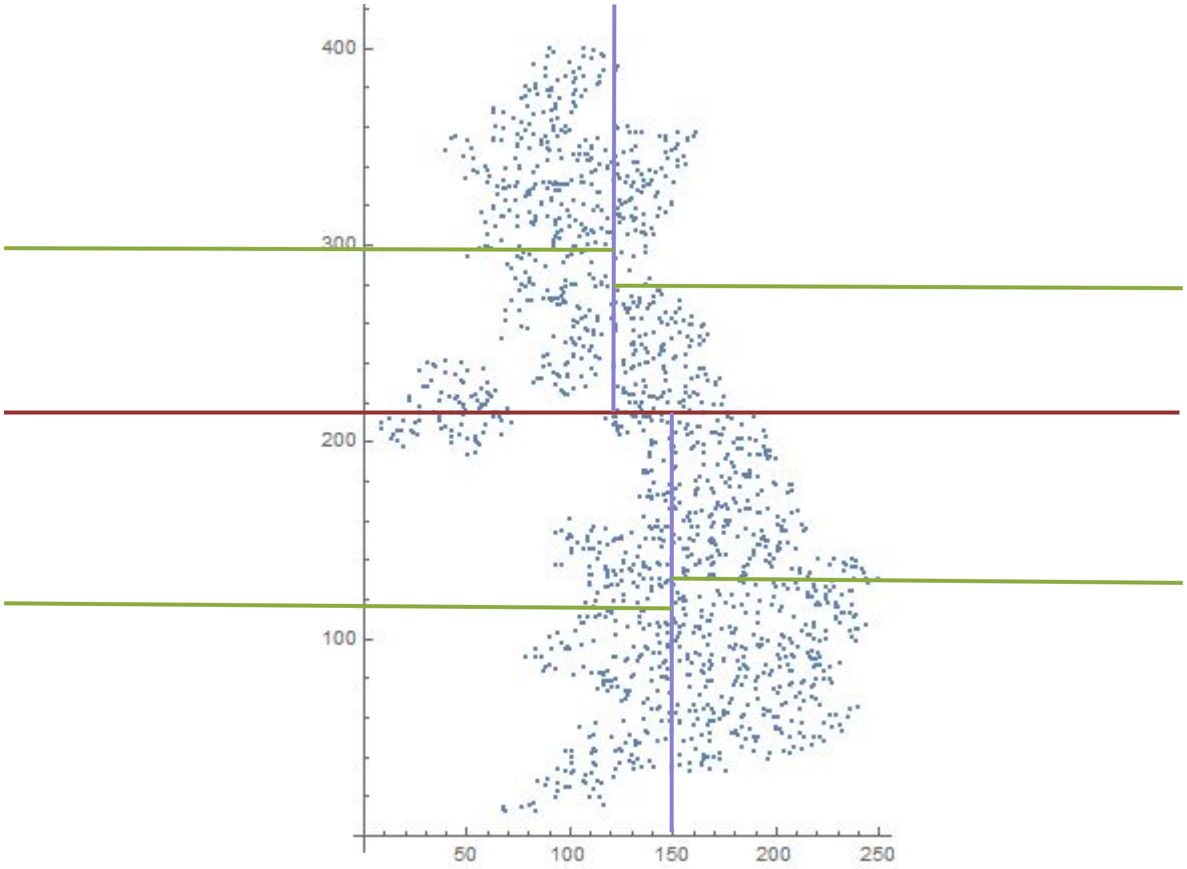
Want to find all people that live in this box.

How many points would we have to search through?

```
[Point1, Point2, Point3, …]
```

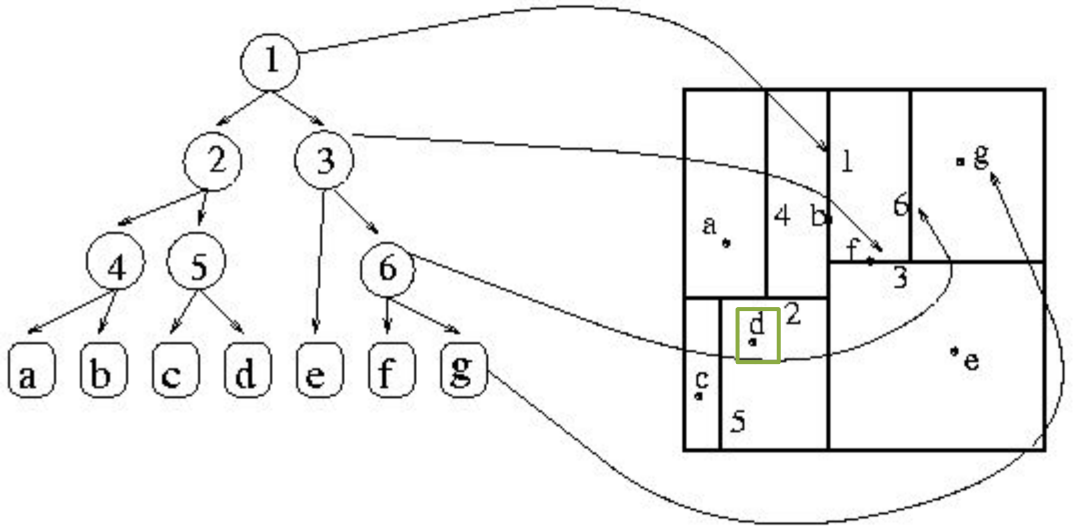


Spatial Index



Spatial Index

This is a tree!



To find the points in a region, just follow the tree

Spatial Index Performance

- Say we are looking for a single Point
- How much work is it if we have n rows?
  - Without a spatial index: O(n)
  - With a spatial index? O(height of tree)
- How tall is the tree? How many times can we divide 1 million points in half?
  - 1,000,000 / 2 / 2 / 2 / 2 / ... / 2 = 1
  - 2^k = 1,000,000
  - k = $\log_2$(1,000,000) ≅ 20
  - This means the lookup is O(log n)
- A million doesn't sound that big. What about a tree of the US?
  - 327.2 million people
  - $\log_2$(327,200,000) ≅ 28
- The US isn't THAT big. What about a tree of China?
  - 1.386 **billion** people
  - $\log_2$(1,386,000,000) ≅ 30

Technicalities

- Now this isn’t for free, the spatial index takes time to build and also takes up extra space
    - O(knlog(n)) where k is the dimension, n is num points
- Indices are great for quickly accessing data, but they can be harder to update
    - It would seem easy to update the tree by adding more points, but we were assuming the tree was balanced.
    - In general, adding an index to your data makes it faster to read but harder to update